# Подключеиние библиотеки

var CroServer = requere(‘CroServer.node’);

CroServer.InitializeMemory(Уникальное имя, размер памяти)

Вызов метода InitializeMemory обязателен до исопльзования любого метода класса CroServer. При использовании других методов до InitializeMemory будет вызвано исключение с текстом «Memory not initialized».

Уникальное имя – строка, используемая для инициализации разделяемой памяти. Размер памяти указывается в МБайтах и должен быть в пределе [64, 1024].

# Класс CroServer

## Синхронные функции класса

### **CroServer.DeSerializeConnection**(сериализованное соединение)

Выполняет десериализацию подключения.

*Входные параметры:*

Сериализованное (строкое) представление соединения, полученное при выполнении метода Serialize экземпляра класса CroConnection.

*Возвращаемое значение:*

Объект класса CroConnection, содержащий информацию о соединении.

### **CroServer.GetVersion**()

Возвращает номер версии в текстовом виде.

*Входные параметры:*

Отсутствуют.

*Возвращаемое значение:*

Строка, содержащая номер версии библиотеки. Библиотека собирается при сборке клиент-сервера, следовательно, имеет тот же номер.

## Асинхронные функции класса

### **Login**(имя пользователя, пароль, хост, таймаут, IP‑адрес, function(err, data) {…});

Осуществляет попытку входа в систему.

*Входные параметры:*

Имя пользователя, пароль, хост – строковые переменные, таймаут – число. IP-адрес – адрес клиента.

*Параметры функции обратного вызова:*

err – ошибка. Всегда одинаковая, описана один раз. Тип: объект. Поля: libraryCode – код ошибки библиотеки, serverCode – код ошибки сервера, text – текст ошибки. Текстовое представление ошибки, например: JSON.stringify(err). Если ошибки нет – null.

data – объект типа CroConnection – сведения о соединении.

### **Logout**(соединение, function(err) {…});

Выход из системы.

*Входные параметры:*

Объект типа CroConnection.

*Параметры функции обратного вызова:*

err – ошибка.

### **AttachBank**(соединение, имя банка (| ИД банка 7.1), function(err, data) {…});

Осуществляет подключение банка по имени для текущей сессии. ВАЖНО! Если при вызове любой функции библиотеки кроме Logout, сессия оказывается закрытой на стороне сервера, функция выполнена не будет (возвращается ошибка 10 из списка ошибок).

*Входные параметры:*

Объект типа CroConnection; имя банка для подключения – строковая переменная.

*Параметры функции обратного вызова:*

err – ошибка.

data – объект класса CroBank.

### **DetachBank**(соединение, function(err) {…});

Отключает банк текущей сессии, при этом закрываются открытые курсоры стороны сервера.

*Входные параметры:*

Объект типа CroConnection.

*Параметры функции обратного вызова:*

err – ошибка.

# Класс CroConnection

## Синхронные функции экземпляра

### **Serialize**();

Выполняет сериализацию соединения.

*Возвращаемое значение:*

Строковое представление свойств соединения для последующего сохранения в сессионном хранилище.

## Члены данных экземпляра

UserName – строка – имя текущего пользователя. Только чтение.

# Класс CroBank

## Синхронные функции экземпляра

### **GetBaseCount**();

Получает число баз.

*Возвращаемое значение:*

Число баз банка

### **GetBase**({номер|мнемокод базы}, по индексу);

Получает базу банка.

*Входные параметры:*

Номер – число, мнемокод базы – строка.

По индексу – указывает, следует (true) интерпретировать номер как индекс базы или (false) как номер базы. По умолчанию – false.

*Возвращаемое значение:*

Объект класса CroBase.

## Асинхронные функции экземпляра

### **CallLuaFunction**(имя функции скрипта, параметры скрипта, function(err, data) {…});

Осуществляет вызов скрипта Lua.

*Входные параметры:*

Имя функции скрипта для выполнения, параметры скрипта – строковые переменные.

*Параметры функции обратного вызова:*

err – ошибка.

data – строка, значение которой получено из скрипта. Возвращается в случае успеха, иначе – null.

### **StringRequest**(текст запроса, уточняющий набор, синхронно, function(err, data) {…});

Выполняет строчный запрос.

*Входные параметры:*

Текст запроса – строка.

Уточняющий набор – объект класса CroRecordSet. Необязательный параметр.

Синхронно – true (по умолчанию) указывает, что функция обратного вызова будет выполнена после выполнения запроса. False – функция обратного вызова будет выполнена после принятия сервером запроса на выполнение.

*Параметры функции обратного вызова:*

err – ошибка.

data – Синхронное выполнение: Объект класса CroRecordSet, содержащий битовую карту отобранных записей. Асинхронное выполнение: идентификатор асинхронной операции. Для получения статуса см CroBank.GetAsyncOperationStatus.

### **GetAsyncOperationStatus**(идентификатор операции, function(err, state, data) {…});

Запрашивает инфморацию об операции.

*Входные параметры:*

Идентификатор асинхронной операции – число.

*Параметры функции обратного вызова:*

err – ошибка.

state – JSON {finishCode – число, указывающее на статус операции. 0 – выполняется, 1 – выполнено, 2 – остановлено, 3 – ошибка. percent - см таблицу ниже.

data: см таблицу ниже.

|  |  |  |
| --- | --- | --- |
| **Операция** | percent | data |
| Строчный запрос | Значение от 0 до 100 | Объект класса CroRecordSet после выполнения операции (finishCode=1), null – в процессе выполнения запроса. |
|  |  |  |

## Члены данных экземпляра

Name – строка – имя подключенного банка. Только чтение.

ID – число – идентификатор подключенного банка. Только чтение.

# Класс CroBase

## Константы класса

CroBase.NoLock – наличие блокировки не проверяется

CroBase.LockNow – проверяется наличие блокировки. Если запись заблокирована, работа метода завершается (никакие действия не выполняются)

CroBase.LockWait – проверяется наличие блокировки. Если запись заблокирована, метод ожидает снятия блокировки, после чего выполняет требуемые действия

## Синхронные функции экземпляра

### **GetFieldCount**();

Получает число полей.

*Возвращаемое значение:*

Число полей базы

### **GetField**(({номер|имя поля}, по индексу);

Получает поле базы.

*Входные параметры:*

Номер – число, имя поля – строка.

По индексу – указывает, следует (true) интерпретировать номер как индекс поля или (false) как номер поля. По умолчанию – false.

*Возвращаемое значение:*

Объект класса CroBaseField

### **CreateRecordSet**();

Возвращает пустой RecordSet, связанный с базой.

*Возвращаемое значение:*

Объект класса CroRecordSet.

## Асинхронные функции экземпляра

### **GetRecord**(системный номер, function(err, data) {…});

Получает запись по системному номеру

*Входные параметры:*

Системный номер - число

*Параметры функции обратного вызова:*

err – ошибка.

data – объект класса CroBaseRecord.

### **LockRecord**(системный номер, немедленно, function(err) {…});

Блокирует запись по системному номеру

*Входные параметры:*

Системный номер – число, немедленно – логическое значение, по умолчанию true.

*Параметры функции обратного вызова:*

err – ошибка.

### **UnlockRecord**(системный номер, function(err) {…});

Снимает блокировку с записи по системному номеру

*Входные параметры:*

Системный номер – число

*Параметры функции обратного вызова:*

err – ошибка.

### **GetRecordSet**(function(err, data) {…});

Получает битовую карту базы.

*Параметры функции обратного вызова:*

err – ошибка.

data – объект класса CroRecordSet.

### **AddRecord**(запись, перечитать, не реплицировать, function(err, data) {…});

Добавляет запись в базу данных.

*Входные параметры:*

Запись – объект класса CroBaseRecord, перечитать, не реплицировать – логическое значение, по умолчанию – false

*Параметры функции обратного вызова:*

err – ошибка.

data – объект класса CroBaseRecord.

### **DeleteRecord**(системный номер, блокировать, корневая связь, function(err) {…});

Добавляет запись в базу данных.

*Входные параметры:*

Системный номер – число. Блокировать запись – логическое значение, по умолчанию – true, корневая связь – логическое значение, по умолчанию false – удаляются связи, имеющие корневую связь.

*Параметры функции обратного вызова:*

err – ошибка.

### **AddLink**(системный номер, поле, связанная база, системный номер в связанной базе, блокировка записи, блокировка связанной записи, не реплиировать, function(err) {…});

Добавляет связь.

*Входные параметры:*

Системный номер – число. Поле – объект класса CroBaseField. Связанная база – объект класса CroBase. Системный номер в связанной базе – число. Блокивки записей – одно из значений констант класса CroBase, по умолчанию – LockNow. Не реплицировать – логической значение, по умолчанию – false.

*Параметры функции обратного вызова:*

err – ошибка.

### **DeleteLink**(системный номер, поле, связанная база, системный номер в связанной базе, блокировка записи, блокировка связанной записи, не реплиировать, function(err) {…});

Разрывает связь.

*Входные параметры:*

Системный номер – число. Поле – имя поля или его номер. Связанная база – номер базы, или мнемокод, или объект CroBase. Системный номер в связанной базе – число. Блокивки записей – одно из значений констант класса CroBase, по умолчанию – LockNow. Не реплицировать – логической значение, по умолчанию – false.

*Параметры функции обратного вызова:*

err – ошибка.

## Члены данных экземпляра

Name – строка – имя базы. Только чтение.

Number – число – номер базы. Только чтение.

Mnemocode – строка – мнемокод базы. Только чтение.

# Класс CroBaseField

## Синхронные функции экземпляра

### **GetLinkedBaseCount**();

Получает число связанных баз.

*Возвращаемое значение:*

Число связанных баз

### **GetLinkedBase**(индекс связанной базы);

Получает связную базу по ее индексу.

*Входные параметры:*

Индекс – число.

*Возвращаемое значение:*

Объект класса CroBase

## Члены данных экземпляра

Name – строка – имя базы. Только чтение.

Number – число – номер базы. Только чтение.

Type – строка – тип поля. Только чтение.

# Класс CroBaseRecord

## Синхронные функции экземпляра

### **GetValuesCount**(поле);

Получает количество значений поля записи.

*Входные параметры:*

Поле – объект класса CroBaseField.

*Возвращаемое значение:*

Число – количество значений.

### **DeleteAllFieldValues**(поле);

### **ClearField**(поле);

Удаляет все значения поля

*Входные параметры:*

Поле – объект класса CroBaseField.

*Возвращаемое значение:*

Отсутствует

### **DeleteFieldValue**(поле, номер значения);

Удаляет значение поля записи

*Входные параметры:*

Поле – объект класса CroBaseField. Номер значения – номер удаляемого поля записи. По умолчанию – 1.

*Возвращаемое значение:*

Отсутствует.

### **IsChanged**();

Проверяет запись на наличие изменений.

*Возвращаемое значение:*

Логическое значение.

## Асинхронные функции экземпляра

### **Lock**(немедленно, function(err) {…});

Блокирует запись.

*Входные параметры:*

Системный номер – число, немедленно – логическое значение, по умолчанию true.

*Параметры функции обратного вызова:*

err – ошибка.

### **Unlock**(function(err) {…});

Снимает блокировку.

*Входные параметры:*

Системный номер – число

*Параметры функции обратного вызова:*

err – ошибка.

### **Update**(перечитать, не реплицировать, function(err) {…});

Обновляет запись.

*Входные параметры:*

Перечитать, не реплицировать – логическое значение, по умолчанию – false

*Параметры функции обратного вызова:*

err – ошибка.

## Комбинированные функции экземпляра

### **GetValue**(Поле, … , function(err, data) {…});

Получает значение поля записи. Поле – объект класса CroBaseField.

1. Простые поля. Синхронный вызов.

*Входные параметры:*

Номер значения – по умолчанию – 1. Если 0 – возвращаются все значения поля в виде массива.

*Параметры функции обратного вызова:*

err – ошибка.

data – полученные значения.

2. Сложные поля. Синхронный вызов.

*Входные параметры:*

База – номер, или мнемокод базы, или объект класса CroBase. *Параметры функции обратного вызова:*

err – ошибка.

data – объект класса CroRecordSet.

3. Словарные поля.

*Входные параметры:*

Номер значения - по умолчанию – 1. Если 0 – возвращаются все значения поля в виде массива. Перекодировать – логическое значение. По умолчанию – true. Номер значения для перекодировки – по умолчанию 1.

Если перекодировать – асинхронный вызов, иначе – синхронный.

*Параметры функции обратного вызова:*

err – ошибка.

data – полученные значения.

4. Поле типа файл.

*Входные параметры:*

Номер значения - по умолчанию – 1. Если 0 – возвращаются все значения поля в виде массива. fileName – определяет, что нужно вернуть. appendFile – дописать файл – по умолчанию false.

Синхронный вызов:

* fileName = “name” – имя файла
* fileName = “ext” – расширение файла
* fileName = “fullname” – имя и расширение файла
* fileName = “pathname” – полное имя файла.

Асинхронный вызов:

* fileName = “” – содержимое файла.
* Иначе fileName – путь, по которому следует запись файл.

### **SetValue**(Поле, … , function(err) {…});

Устанавливает значение поля записи. Поле – объект класса CroBaseField.

1. Простые поля. Синхронный вызов.

*Входные параметры:*

Значение (Массив значений) для установки. Номер значения – по умолчанию – 1. Если передан массив значений – он заменит все значения поля. Если номер значения лежит в диапазоне от 1 до числа значений поля – значение поля будет заменено, иначе – добавлено. Если в качестве значения в функцию передано пустое значение – значения поля будет удалено.

*Параметры функции обратного вызова:*

err – ошибка.

2. Поле типа файл. Асинхронный вызов.

*Входные параметры:*

Путь к файлу или содержимое файла (\*). Номер значения – по умолчанию – 1. Имя файла. Расширение файла. realFile – логическое выражение.

Если realFile = true – (\*) рассматривается как путь к файлу, иначе – как содержимое.

*Параметры функции обратного вызова:*

err – ошибка.

## Члены данных экземпляра

SN – число – системный номер записи. Только чтение.

Base – объект класса CroBase - база записи.

# Класс CroRecordSet

## Константы класса

CroRecordSet.ClientSide – курсор стороны клиента. При передаче этого значения курсор стороны сервера не открывается, запрос всегда выполняется, возвращаются актуальные данные на момент запроса.

CroRecordSet.ServerSideOpenAlways – курсор стороны сервера. При передаче этого значения курсор всегда переоткрывается, запрос выполняется принудительно.

CroRecordSet.ServerSideOpenOrCreate – курсор стороны сервера. При передаче этого значения проверяется наличие открытого курсора для текущей сессии. Если открытый курсор, соответствующий запросу, найден, то запрос не выполняется, клиенту возвращается результат без физического выполнения запроса. Если открытый курсор не найден, то запрос выполняется, и открывается курсор стороны сервера, соответствующий тексту запроса.

## Синхронные функции экземпляра

### **GetSn**(индекс);

Получает системный номер по индексу.

*Входные параметры:*

Индекс – число.

*Возвращаемое значение:*

Системный номер.

### **SetSn**(системный номер);

Устанавливает системный номер

*Входные параметры:*

Системный номер – число.

*Возвращаемое значение:*

Отсутствует.

### **ClearSn**(системный номер);

Удаляет системный номер

*Входные параметры:*

Системный номер – число.

*Возвращаемое значение:*

Отсутствует.

### **TestSn**(системный номер);

Проверяет, есть ли номер в битовой карте.

*Входные параметры:*

Системный номер – число.

*Возвращаемое значение:*

Логическое значение.

## Асинхронные функции экземпляра

### **GetRecord**(системный номер, function(err, data) {…});

Получает запись по системному номеру.

*Входные параметры:*

Системный номер - число

*Параметры функции обратного вызова:*

err – ошибка.

data – объект класса CroBaseRecord.

### **GetRecordByIndex**(индекс, function(err, data) {…});

Получает запись по индексу из битовой карты.

*Входные параметры:*

индекс – порядковый индекс (начиная с 1).

*Параметры функции обратного вызова:*

err – ошибка.

data – объект класса CroBaseRecord.

### **CountAsync**(function(err, data) {…});

Получает количество системный номеров.

*Входные параметры:*

*Параметры функции обратного вызова:*

err – ошибка.

data – количество системный номеров.

## Члены данных экземпляра

Count – число – количество системный номеров. Только чтение.

Base – объект класса CroBase - база записи.

# Коды ошибок библиотеки (libraryCode)

|  |  |  |
| --- | --- | --- |
| **Название** | **Код** | **Пояснение** |
| WrongInputArguments | 1 | Неверные входные параметры функции |
| InternalException | 2 | Внутренняя ошибка сервера: см. serverCode и text |
| NotEqualProtocol | 3 | Нессответствие протокола клиента и сервера |
| NotEqualSerialNumber | 4 | Несоответствие серийных номеров клиента и сервера |
| NotEqualBuild | 5 | Несоответствие сборок клиента и сервера |
| DisableUser | 6 | Заблокированный пользователь |
| NotChangedPass | 7 | Пользователь должен был изменить пароль |
| DefaultLogonError | 8 | Прочая ошибка входа: см. serverCode и text |
| LibrarySessionNotFound | 9 | Сессия не найдена по идентификатору на стороне библиотеки |
| ServerSessionNotFound | 10 | Сессия не найдена на стороне сервера (завершилась по таймауту) |
| BankNotAttached | 11 | К текущей сессии не присоединен банк |
| --- | 12 | Внутренняя ошибка библиотеки |
| --- | 13 | Внутренняя ошибка библиотеки |
| SessionCountExceeded | 14 | Превышено число сессий пользователя |
| InterprocessMemoryException | 15 | Нехватка разделяемой памяти |